Team #13

Voting System

Software Design Document

Name (s): Leo Dong, Alex Johnson, Janani Kannan, Ashwin Wariar

Date: (03/01/2024)

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to describe the system design of a voting system for elections that has two different election processes: Closed Party List (CPL) and Open Party List (OPL). This document will highlight the design of the system, which is described using the Unified Modeling Language (UML), as well as other explanations.

1.2 Scope

The voting system described in this document can be used to calculate the results of any type of election that utilizes Open or Closed Party Listing. The United States of America is an example of a country where many states utilize Open or Closed Party Listing rules in their primaries. The system will be used by election officials to count ballots, and it will calculate the winner based on the requirements of the election using the counted votes. Any government entity or organization that wants to hold fair elections can use this voting system.

1.3 Overview

Section 2 is the system overview which details a description of the system and how the design reflects it. The UML diagrams are included in Section 3 to illustrate the system architecture. Section 4 goes over a description of the data used in the system, as well as important definitions for the data. Section 5 highlights the component design of the system, and Section 6 shows how users interact with the system through diagrams and pictures. Finally, Section 7 details a requirements matrix that shows how the use cases are used in the system.

1.4 Reference Material

SRS document:
https://github.umn.edu/umn-csci-5801-02-S24/repo-Team13/blob/855111c97004a2fed6ff11aeab6b97fb881f1e24/SRS/CSci%205801%20-%20SRS_Team13.pdf

1.5 Definitions and Acronyms

CPL - Closed Party List; A type of voting system where voters express their preference for a political party/affiliation

OPL - Open Party List; A type of voting system where voters express their preference for a particular candidate

UML - Unified Modeling Language; A visual specification language for describing, specifying, and constructing software systems.

2. System Overview

2.1 Context

For our Voting System project, we are building a standalone software that processes ballots from either an OPL or a CPL election and produces relevant information about the results of the election. Stakeholders in the system include election officials and other users of this system. Those that may view the results of the election as calculated by this system may also be considered stakeholders, here election officials and audit testers.

2.2 Functionality

The program takes as input the filename of a CSV file containing ballots. The user can provide inputs through the Command Line Interface (CLI), where they type in a filename on a command prompt or a terminal, or the Graphical User Interface (GUI) where they are able to select a file from their local directory via a "browse" option.

The CSV file is processed; relevant parts of the file such as the headers and rows of ballot information are extracted.

3. System Architecture

3.1 Architectural Design

Modules & associated classes from UML:

- **Voting System:** VotingSystem, OPL, CPL, Electable, Candidate, Party
  - VotingSystem is an abstract class that the OPL or CPL election types inherit general methods from, providing methods and attributes used in either election type
  - Electable is an abstract class providing attributes and methods for objects that may receive votes and may win elections
  - Candidate is the votable object in OPL, a person
  - Party is the votable object in CPL, a party made up of people who cannot individually receive votes
- **User Interface:** Window
  - Governs user input of the filename containing ballot information
  - Manages operations to display election results in a pre-specified format
- **File Operations:** FileOps, Audit, RawData, Parser, Header
  - Utility classes for working with and creating file objects
  - Audit is a singleton class, allowing it to be treated similarly to a log file
  - RawData is the result of parsing the ballots .csv file
  - Audit and RawData inherit from the FileOps class, which may also be instantiated to work with a generic file
  - Parser parses the ballot file, as dictated by Header's election type
  - Header consumes the file header to determine election type and other information
- **Results Computation:** Electable, Candidate, Party, OPL, CPL

- OPL and CPL realize the algorithm for calculating election results
- OPL or CPL is created based on the file header parsed in Parser and Header

Relationships between modules

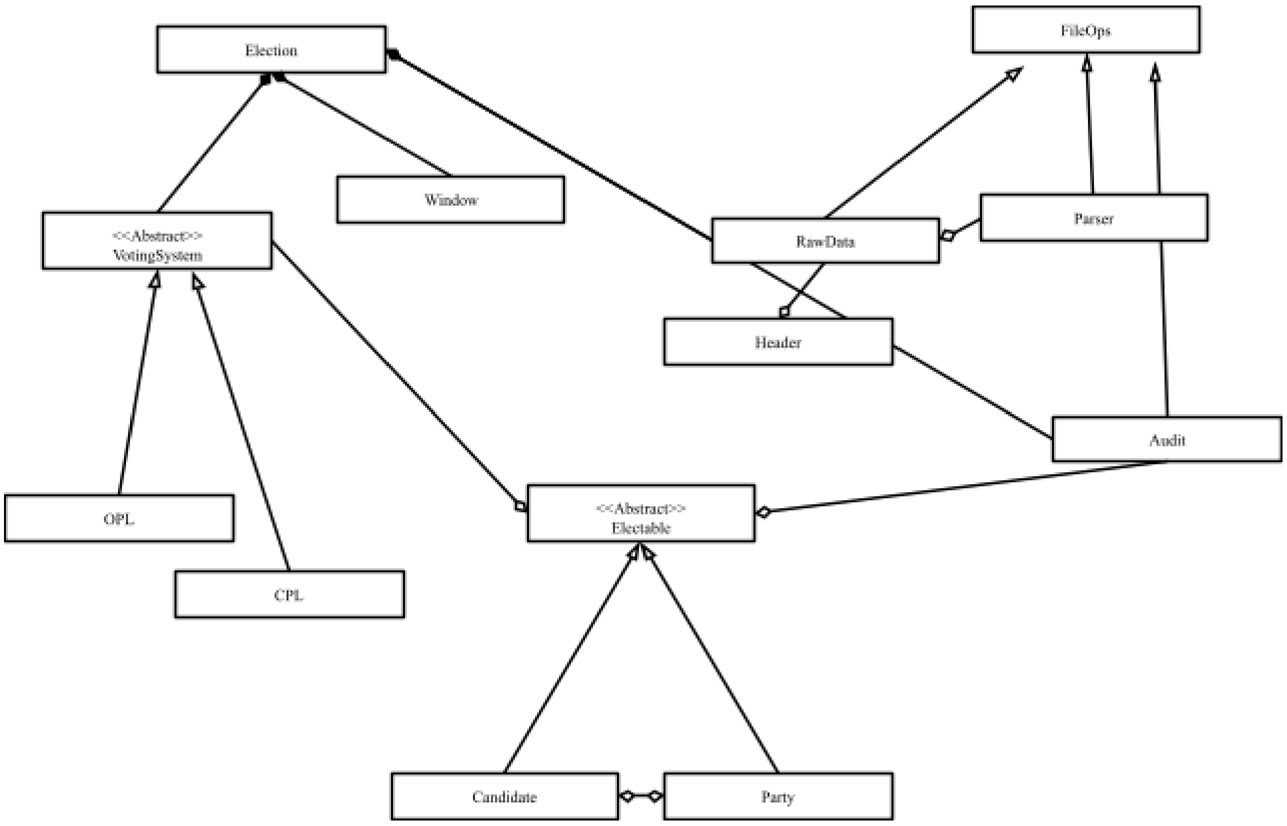


Figure 3.1.1. Overview UML diagram of system design and class relationships. For more information, please see the more detailed UML diagrams in section 4.

Activity diagram:

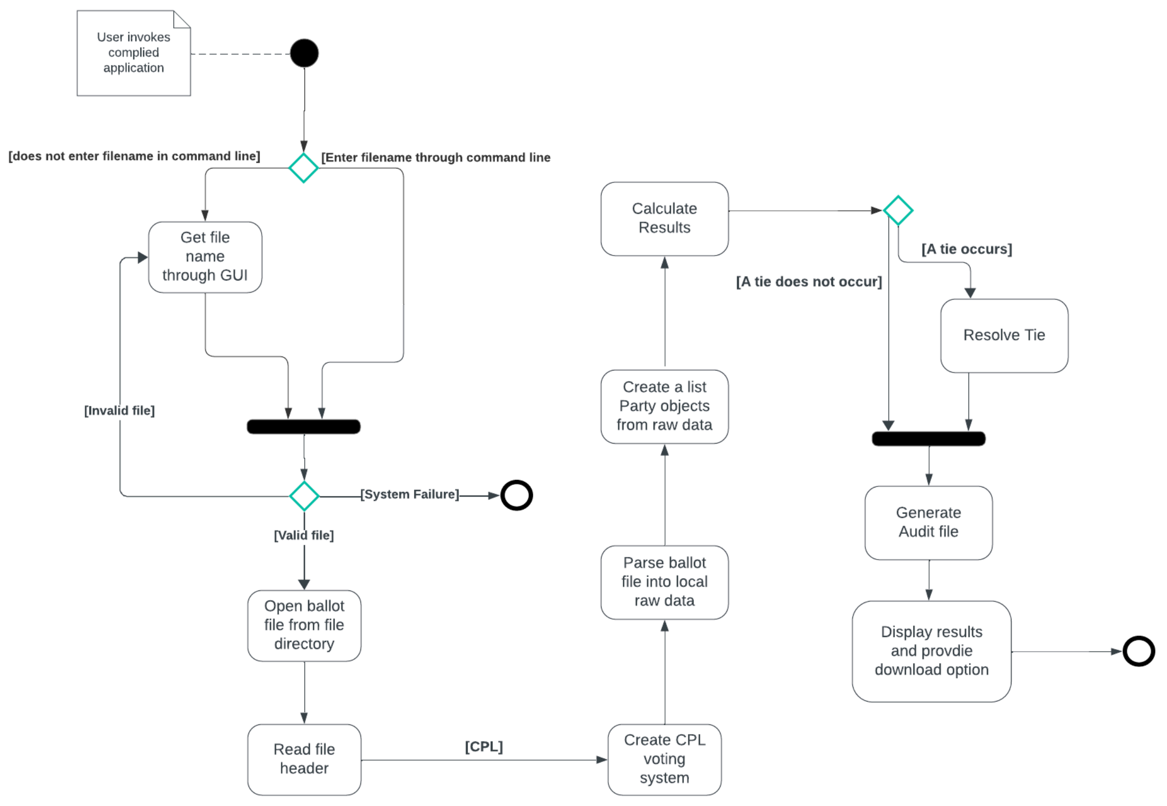

User invokes complied application
[does not enter filename in command line]
[Enter filename through command line
Get file name through GUI
[Invalid file]
[System Failure]
[Valid file]
Open ballot file from file directory
Read file header
[CPL]
Create CPL voting system
Parse ballot file into local raw data
Create a list Party objects from raw data
Calculate Results
[A tie occurs]
[A tie does not occur]
Resolve Tie
Generate Audit file
Display results and provdie download option

Figure 3.1.1. CPL Activity Diagram

3.2 Decomposition Description

Top-level data flow diagram:

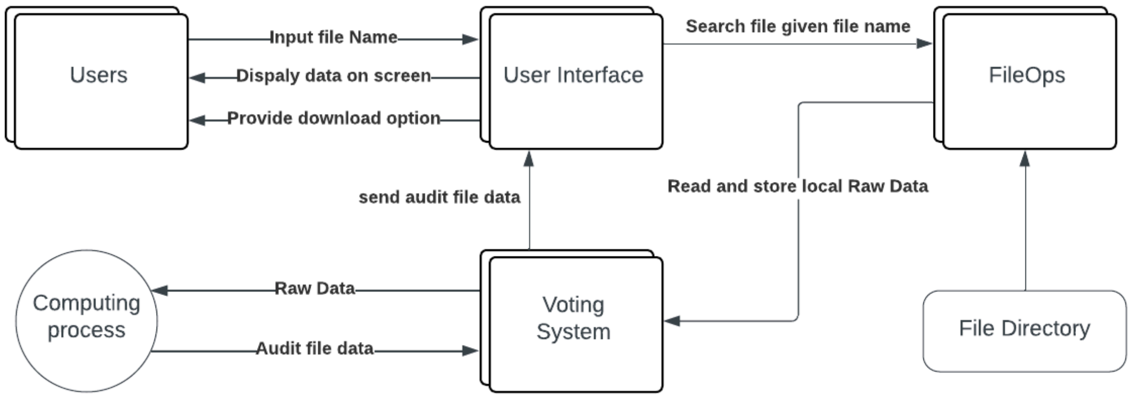


Figure 3.2.1. Data flow diagram.

Structural decomposition diagram (User Interface):

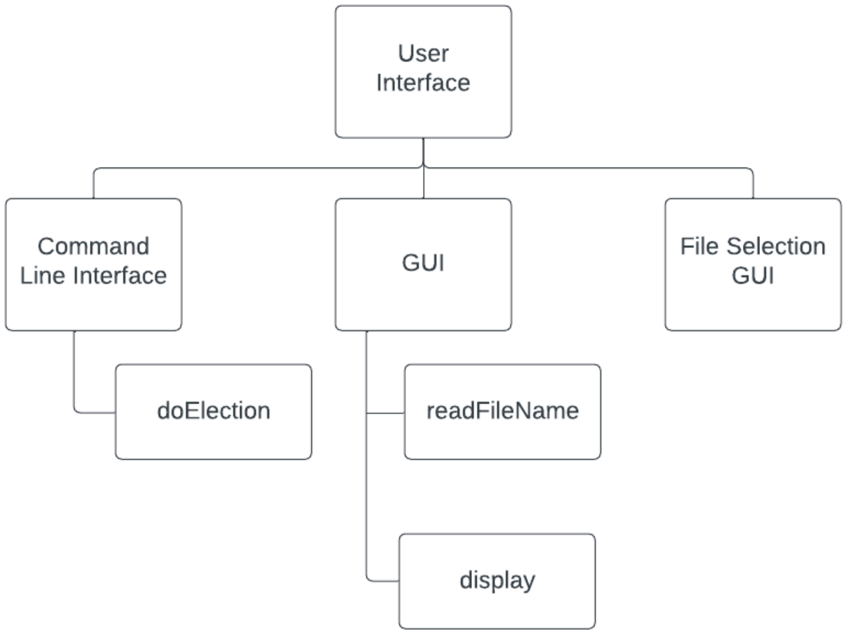


Figure 3.2.2. User Interface structural decomposition diagram

Structural decomposition diagram (FileOps):

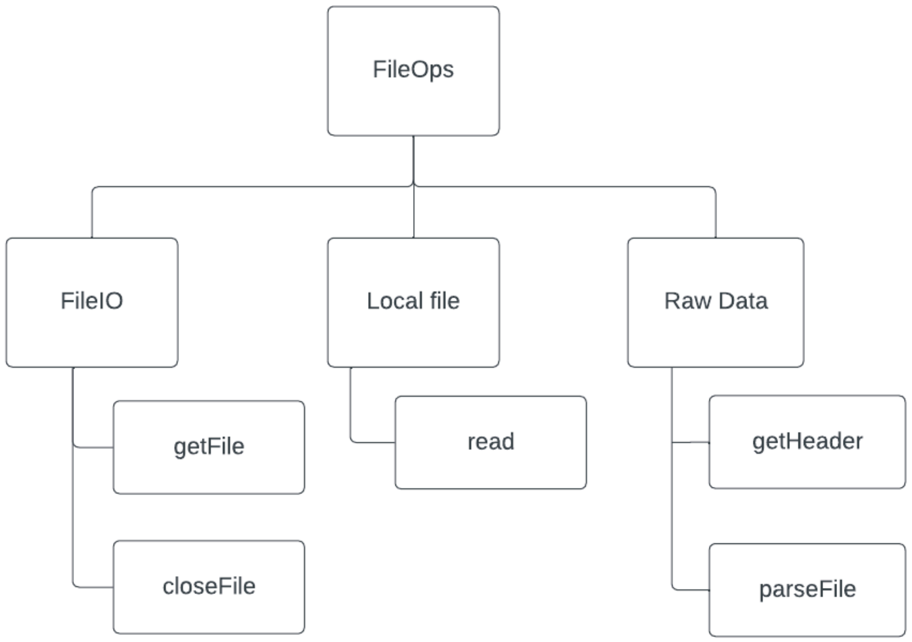


Figure 3.2.3. FileOps structural decomposition diagram

Structural decomposition diagram (VotingSystem):

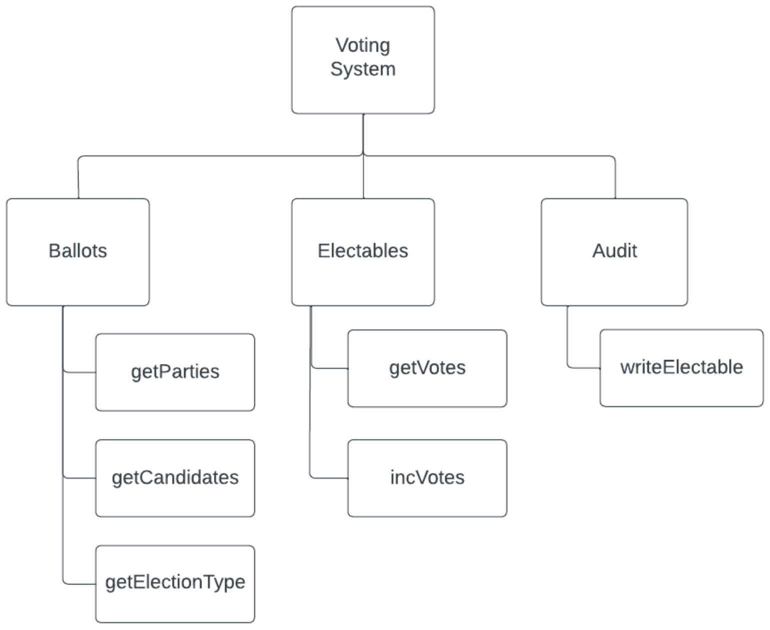


Figure 3.2.4. VotingSystem structural decomposition diagram

Structural decomposition diagram (Calculating Process):

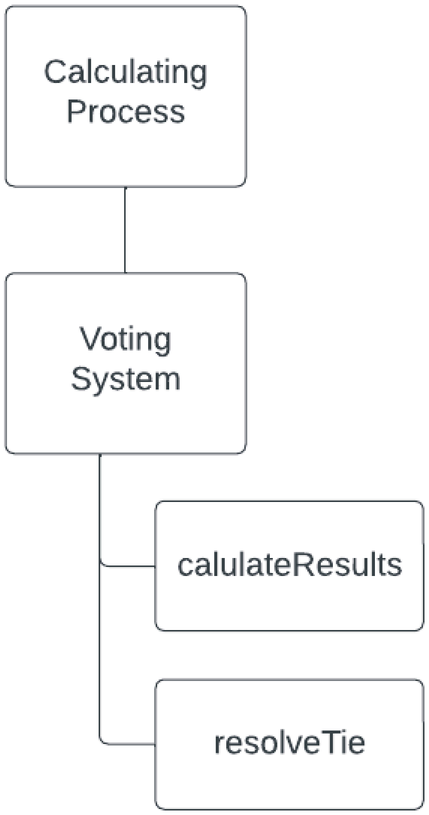


Figure 3.2.5. CalculatingProcess structural decomposition diagram

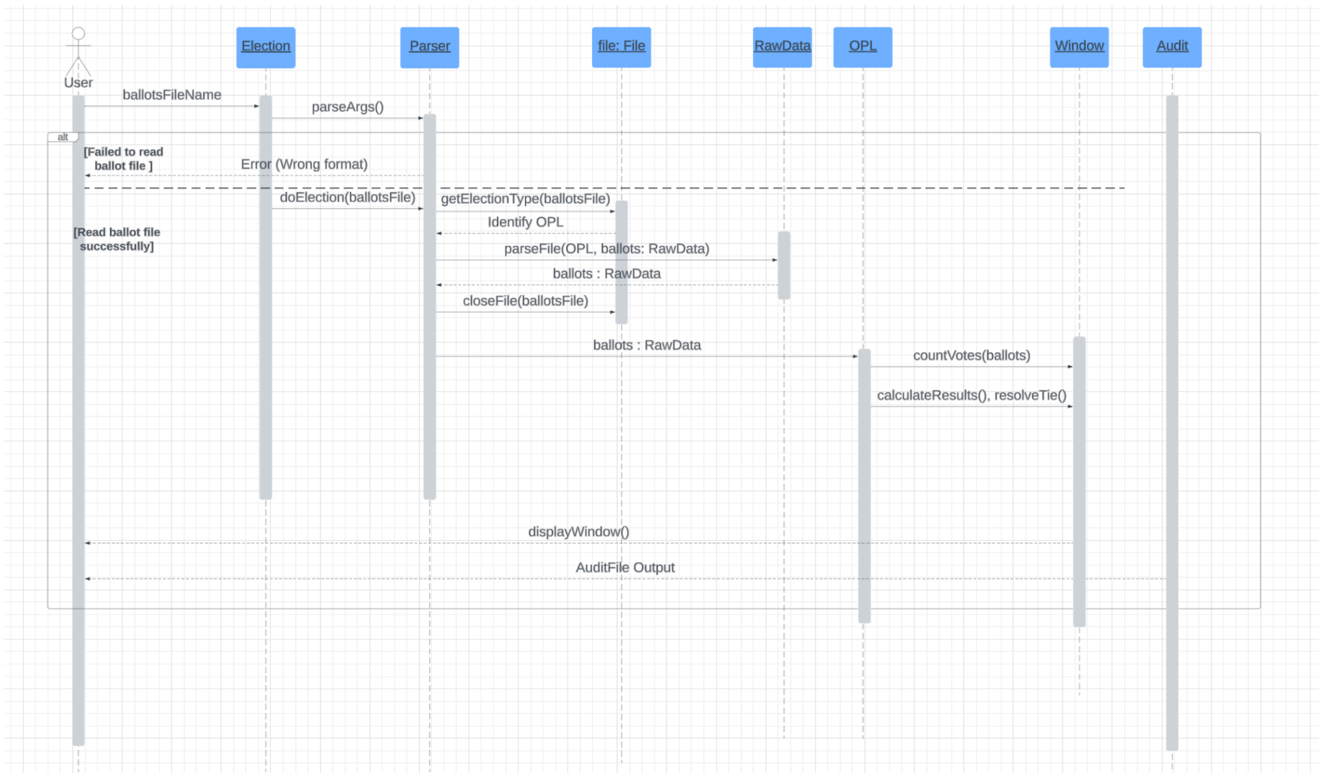


Figure 3.2.6. OPL sequence diagram

<u>Outline of the OPL Sequence Diagram</u>

- ballotsFileName is parsed and doElection() is called with the filename
  - **Exception:** ballotsFileName could not be parsed; Failed to read ballot file
- getElectionType() is called by Parser and it identifies election type as OPL
- The file is parsed, and ballots, a RawData object, is populated with all the necessary fields, including headerInfo, ballotInfo and partyCount
- ballots, a RawData object, is passed to OPL (realizes from VotingSystem)
- OPL calls countVotes()
  - Reads each line of ballot information and increments candidate “votes” field according to the position of ballot as well as the party “votes” field for the corresponding party of the candidate
- OPL calls calculateResults()
  - Allocates seats using the Largest Remainder Approach to compute seat allocations **and** tie resolution when there is a tie between two candidates -> call resolveTie()
- Display results
- Produce Audit File

3.3 Design Rationale

The main program is run from the Election class. This allows the program to create a GUI input window, if needed, to handle GUI input parsing, and, eventually, to display the results. It may be the case that future users wish to change how the file enters the system. As such, this is done via a separate Window module. The Election class creates the voting system after parsing the input file. This input file may be of a different form or from a different source in the future, so parsing the input ballots is done by the FileOps class and children classes. Additionally, different systems may handle files differently, so this FileOps class may function as a sort of API to the native system file calls. In order to standardize the election data a RawData class is created, with a Header class to standardize election information. A singleton Audit class is created, allowing future audit format to be easily changed or swapped out. Similarly, different election types may be added, which may involve different voting systems or rules and for different objects than Candidates or Parties. As such, an overarching Electable type and VotingSystem was created. The specific election result calculation process is done differently by individual instances of the VotingSystem class through polymorphism. This method significantly improves code reuse and provides a single entry for special elections during the different times of the year.

4. Data Design

For greater clarity, UML diagrams are broken down into parts below. Each diagram focuses on illustrating the structure of a group of classes.

<u>UML Diagram #1 focuses on:</u>

Election, <<Abstract>> VotingSystem, OPL, CPL, <<Abstract>> Electable, Candidate, Party

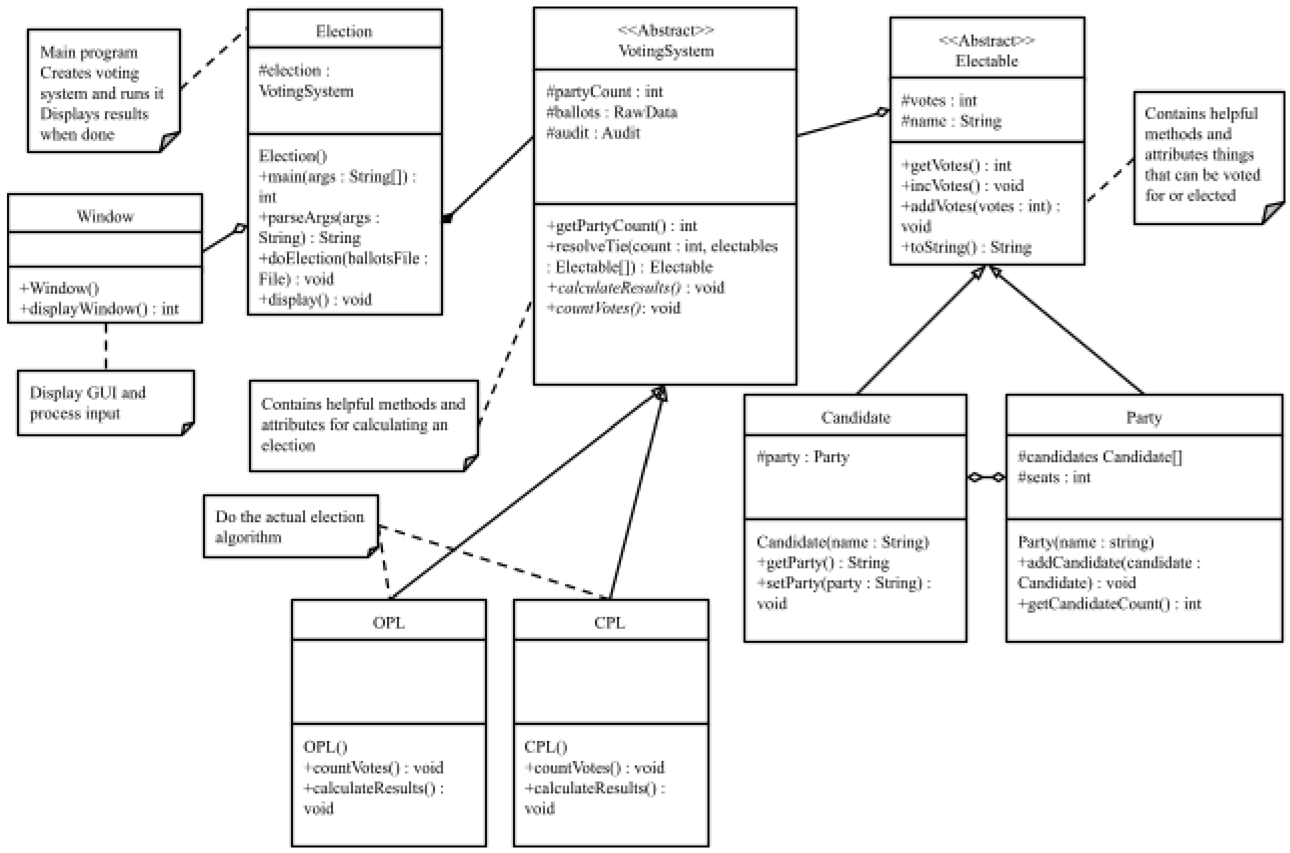

Main program Creates voting system and runs it Displays results when done
Election
#election : VotingSystem
Election()
+main(args : String[]) : int
+parseArgs(args : String) : String
+doElection(ballotsFile : File) : void
+display() : void
Window
+Window()
+displayWindow() : int
Display GUI and process input
<<Abstract>> VotingSystem
#partyCount : int
#ballots : RawData
#audit : Audit
+getPartyCount() : int
+resolveTie(count : int, electables : Electable[]) : Electable
+calculateResults() : void
+countVotes(): void
Contains helpful methods and attributes for calculating an election
Do the actual election algorithm
OPL
OPL()
+countVotes() : void
+calculateResults() : void
CPL
CPL()
+countVotes() : void
+calculateResults() : void
<<Abstract>> Electable
#votes : int
#name : String
+getVotes() : int
+incVotes() : void
+addVotes(votes : int) : void
+toString() : String
Contains helpful methods and attributes things that can be voted for or elected
Candidate
#party : Party
Candidate(name : String)
+getParty() : String
+setParty(party : String) : void
Party
#candidates Candidate[]
#seats : int
Party(name : string)
+addCandidate(candidate : Candidate) : void
+getCandidateCount() : int

Figure 4.0.1. Election and VotingSystem specific UML diagram

UML Diagram #2 focuses on:

Audit, FileOps, RawData, Header, Parser

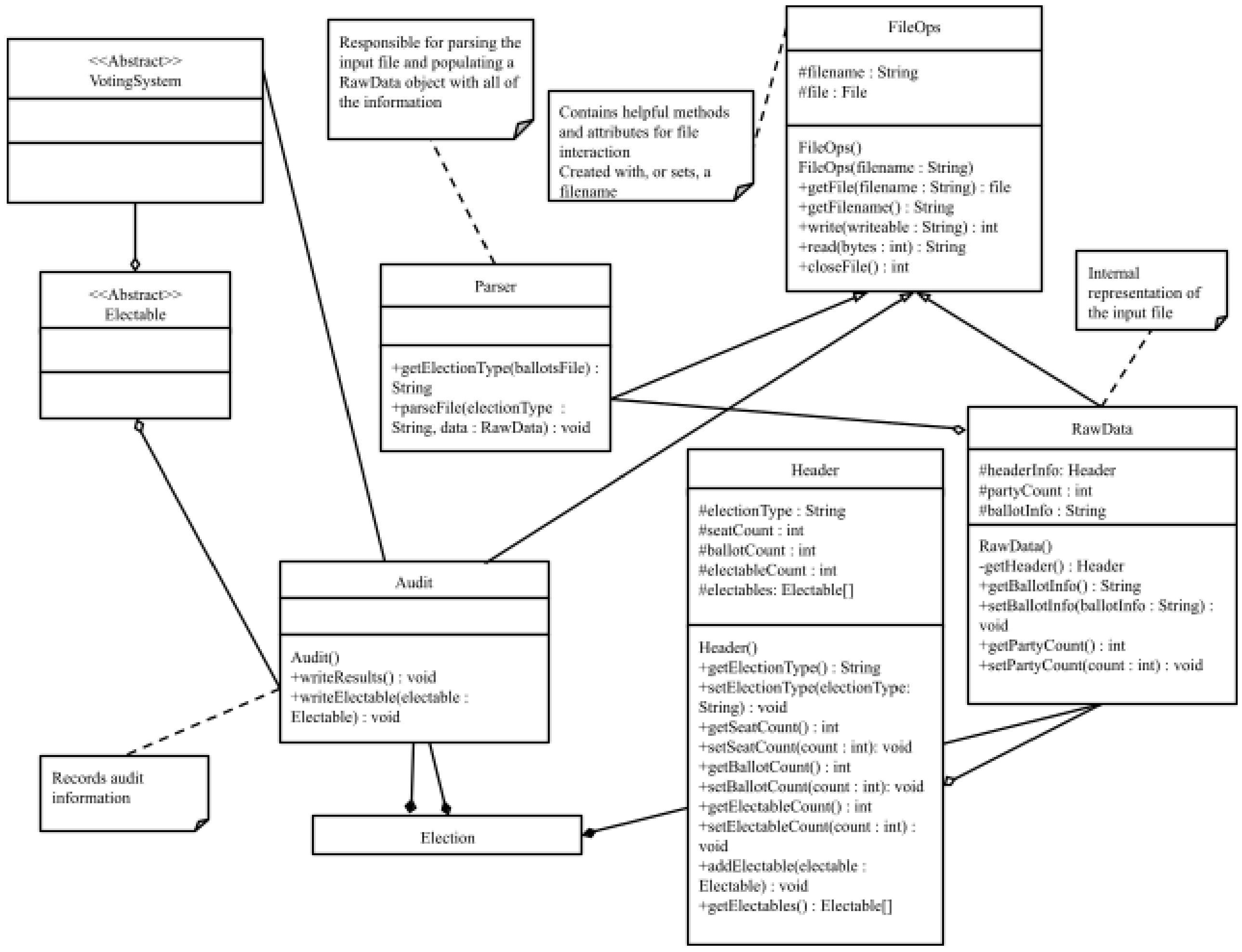

<<Abstract>>
VotingSystem
Responsible for parsing the input file and populating a RawData object with all of the information
Contains helpful methods and attributes for file interaction
Created with, or sets, a filename
FileOps
#filename : String
#file : File
FileOps()
FileOps(filename : String)
+getFile(filename : String) : file
+getFilename() : String
+write(writeable : String) : int
+read(bytes : int) : String
+closeFile() : int
<<Abstract>>
Electable
Parser
+getElectionType(ballotsFile) : String
+parseFile(electionType : String, data : RawData) : void
Internal representation of the input file
RawData
#headerInfo: Header
#partyCount : int
#ballotInfo : String
RawData()
-getHeader() : Header
+getBallotInfo() : String
+setBallotInfo(ballotInfo : String) : void
+getPartyCount() : int
+setPartyCount(count : int) : void
Header
#electionType : String
#seatCount : int
#ballotCount : int
#electableCount : int
#electables: Electable[]
Header()
+getElectionType() : String
+setElectionType(electionType: String) : void
+getSeatCount() : int
+setSeatCount(count : int): void
+getBallotCount() : int
+setBallotCount(count : int): void
+getElectableCount() : int
+setElectableCount(count : int) : void
+addElectable(electable : Electable) : void
+getElectables() : Electable[]
Audit
Audit()
+writeResults() : void
+writeElectable(electable : Electable) : void
Records audit information
Election

Figure 4.0.2. File parsing specific UML diagram

4.1 Data Description

The system uses a combination of objects and arrays to represent and organize the relevant data being parsed from the input file. The processing of the objects and other data is done by algorithms in the Parser class which exports a RawData object including information in the file's header as well as the ballot information. The CPL or OPL classes calculate election winners. There is no major database that is being used for the voting system.

Electable candidates and their parties are stored as 'electables', a list of Electable objects in the Header class. RawData objects have a headerInfo attribute, through which all header information can be accessed. These Electable objects have attributes related to their human-recognizable names, the number of votes they have received, with methods to update or retrieve the number of votes. There is also a method, used by the Audit class, to provide a brief summary or description of the electable object.

For an Open Party List election, the system only has to worry about the candidate the voters select, and can use the ‘electables’ attribute to map the candidate to the party. The votes for each candidate’s associated party are also constantly updated for use in calculating results later on using the Largest Remainder approach. In a Closed Party List election, the system has to be concerned with the party, and can then simply choose the candidates based off of their party rank, which is predetermined – each party has an array of Candidates stored in order of their rank.

4.2 Data Dictionary

Candidate

Data Name	**Data Type**	**Description**
party	Party	A Party object that represents a certain party

Electable

Data Name	**Data Type**	**Description**
votes	int	Represents the number of votes cast in the whole election
name	string	A string that represents either the name of a candidate (OPL) or the name of a party (CPL)

Election

Data Name	**Data Type**	**Description**
election	VotingSystem	The entirety of the election is in a VotingSystem object for the algorithm to run it on

FileOps

Data Name	Data Type	Description
filename	string	Represents the name of the file being passed in
file	File	Represents the actual file itself being passed in

Header

Data Name	Data Type	Description
electionType	string	Represents the type of election (either CPL or OPL)
seatCount	int	Represents the number of seats available in an election
ballotCount	int	Represents the number of ballots cast for a certain election
electableCount	int	Either the number of parties or the number of candidates depending on the type of election being run
electables	Electable[]	A list of Electable object that contains all the parties and all of the candidates in each party

Party

Data Name	Data Type	Description
candidates	Candidate[]	A list of all the candidates in a particular party
seats	int	The number of seats allocated to a party

RawData

Data Name	Data Type	Description
headerInfo	Header	A Header object that contains the relevant file header information
partyCount	int	A number that is the number of parties in an election
ballots	String	All of the preferences for the candidates are contained in one string

Voting System

Data Name	Data Type	Description
partyCount	int	The number of parties in the election
ballots	RawData	All of the ballots from the election are contained in a RawData object to be parsed
audit	Audit	The audit file is contained in an Audit object to decide the type of election, so the specific algorithm can be run

5. Component Design

5.1 User Interface

This system behaves as a middleware between the main Voting System and the FileOps System. It invokes the FileIO System to import ballot files given user input through the command line or GUI. Additionally, it supports the display of final election results and provides a download audit file option for the users.

Functions:

doElection()
This function is called through the command line interface when the complied application is invoked. It detects whether an additional argument is passed into the program.

Input: An optional list of strings, `args.`
Algorithm:
1. If a filename is found within `args`:
    a. Send the filename to the FileOps System.
2. Else:
    a. Prompt the user for a filename through GUI

Output: No output

readFileName()
This function serves as reading the data (filename) in a text field that is provided for user input

Input: A string representing the filename.
Algorithm:
1. Read the designated text field
2. The program proceeds to the FileOps System.

Output: No output

displayWindow()
This function functions as displaying the election results in the audit file on the screen; meanwhile, the download button is activated with a downloadable audit file for the user.

Input: No input
Algorithm:
1. Access the `audit` attribute in the `Voting System` class
2. Display the data on the screen.
3. Convert the data into a binary file format for downloading.
4. Activate the download button, enabling the user to download the audit file.

Output: No output

5.2 FileOps System

This component functions as a subsystem to read the ballot file given a filename with an error check for the validity of the ballot file and parse the file data into RawData Objects for further processing. By reading the file header, It determines and creates a specific instance for the Voting System.

Functions:

getFile()
This function works by getting the ballot file from the file directory with necessary validation checking.

Input: A string, `filename`
Algorithm:

1. Search the file directory for this filename.
    a. If the file does not exist, return an error message.
    b. Else, check the file format of the file.
        i. If the file format is not valid, return an error message.
        ii. Else, return the file

Output: A file corresponding to the `filename`

read()

This function reads the file object given a number of bytes

Input: An integer representing the number of bytes to read, `bytes`

Algorithm:

1. Read the file binary data until the number of bytes is reached.

Output: A string containing the binary data.

parseFile()

This function parses the string data read from ballot files into an instance of the `RawData` class for efficient data processing.

Input: A string representing election type, `electionType` and a RawData object, `data`

Algorithm:

1. If `type` is OPL, read the file in OPL format and update the attributes of a `RawData` instance.
2. If `type` is CPL, read the file in CPL format and update the attributes of a `RawData` instance.
3. Else, return an error message.

Output: No output.

5.3 Voting System

This component serves as the primary program for the intended users. It is composed of raw input data from ballot files, utilizes an intermediate data structure for the Calculating Process, and outputs election results in an audit file format.

Functions:

getParties()

This function retrieves a list of strings representing all parties involved in an election from a `RawData` class object. It then converts this list into a list of `Party` class objects within the Voting System for local data storage. The function is invoked on a `RawData` class object.

Input: No input.

Algorithm:

1. Access and return the ‘parties’ attribute from the instances of the `RawData` class.

Output: The ‘parties’ attribute.

getCandidates()

This function retrieves a list of strings representing all candidates involved in an election from a `RawData` class object. It then converts this list into a list of `Candidate` class objects within the Voting System for local data storage. The function is invoked on a `RawData` class object

Input: No input

Algorithm:

1. Access and return the `candidates` attribute from the instances of the `RawData` class.

Output: the `candidates` attribute.

getVotes()

This function acts as a getter for the `votes` attribute within the abstract `Electable` class. The realizations of the `Electable` class, the `Candidate` and `Party` class, can call this method to retrieve the total vote count for a candidate or party object, respectively.

Input: No input

Algorithm:

1. Access and return the `votes` attribute from the instances of the `Candidate` or ‘Party` class.

Output: the `votes` attribute

incVotes()

This function serves the purpose of incrementing the value of the `votes` attribute in the abstract `Electable` class. It behaves as counting the ballot votes for a candidate or party during the voting process. This method is called upon objects of the `Candidate` and `Party` classes.

Input: No input.

Algorithm:

1. Adds ‘1’ to the value of the `votes’ attribute.

Output: No output.

writeElectable()

This function is designed to write sections of the election result into an audit file object and store it in the program for the user to download.

Input: A string

Algorithm:

1. Writes the string to the `Audit` class object stored in `VotingSystem`

Output: No output

5.4 Calculating Process

This component acts as a subsystem designed to manage the algorithm for counting election ballots and computing the results, accommodating various instances of Voting System through polymorphism.

Functions:

countVotes()

The program takes the `ballots` string from a RawData object and counts votes

Input: A RawData object

Algorithm:

1. Iterate through `ballots` string one line after another
    a. According to the position of the '1', increment the `votes` field of Candidate as well as their associated party if OPL or only the `votes` field of the party if CPL

Output: No Output

calculateResults()

The program iterates through the `ballots` attribute of either a `CPL` or `OPL` class object. This iteration facilitates the gathering of necessary information to fulfill the `audit` attribute.

Input: A realized `VotingSystem` object.

Algorithm:

1. getPartyCount() is invoked on the `ballots` attribute to retrieve the number of parties and write to the audit file
2. getBallotCount() is invoked on the `ballots` attribute to receive the number of ballots.
3. Iterates through the list of `Electable` class objects in the `headerInfo` attribute of `ballots`:
    a. If it is a `Candidate` class object, retrieve the candidate's name and its party affiliation by calling toString() and getParty(), respectively.
    b. If it is a `Party` class object, retrieve the party's vote count by calling getVotes(). Then, use the largest remainder approach to allocate seats. The remaining votes are calculated by applying the modulus operator to the party's vote count with the number of the First Allocation of Seats for that party.
4. The program assigns a Second Allocation of Seats for all parties based on the descending ranking of the remaining votes list.
5. The Final Seat Total for each party is calculated by adding the First Allocation of Seats and the Second Allocation of Seats.
6. The ratio of the number of votes to the number of seats of a party is calculated by dividing the percentage of the party votes in terms of total ballots divided by the percentage of seats in terms of the total seats.
7. The number of seat winners is determined by the Final Seat Total of each party
    a. If the election type is Open party, each seat winner of the party is further determined by the descending order of candidates' vote count.
    b. If the election type is Closed party, assign each seat winner

of the party to the default order of candidates in that party.

Output: No output

resolveTie()

The function is designated to resolve any tie situation that occurs during the calculation process, either determining which party wins the Second Allocation of Seats when two or more parties have the same remaining vote counts or determining the seat winner for a candidate when two or more candidates have the same vote counts in OPL voting system.

Input: An integer representing the number of tied positions, `num_of_tied,` and a list of `Electable` class objects that are in the tie, `ties.`

Algorithm:

1. Assign a value margin ranging from 0 to 1 to each object of the `Electable` class in the list.
2. The program simulates a tiebreaker by randomly generating a decimal number that falls within one of these value margins.
3. Determine the instance of the `Electables` class having the value margin.

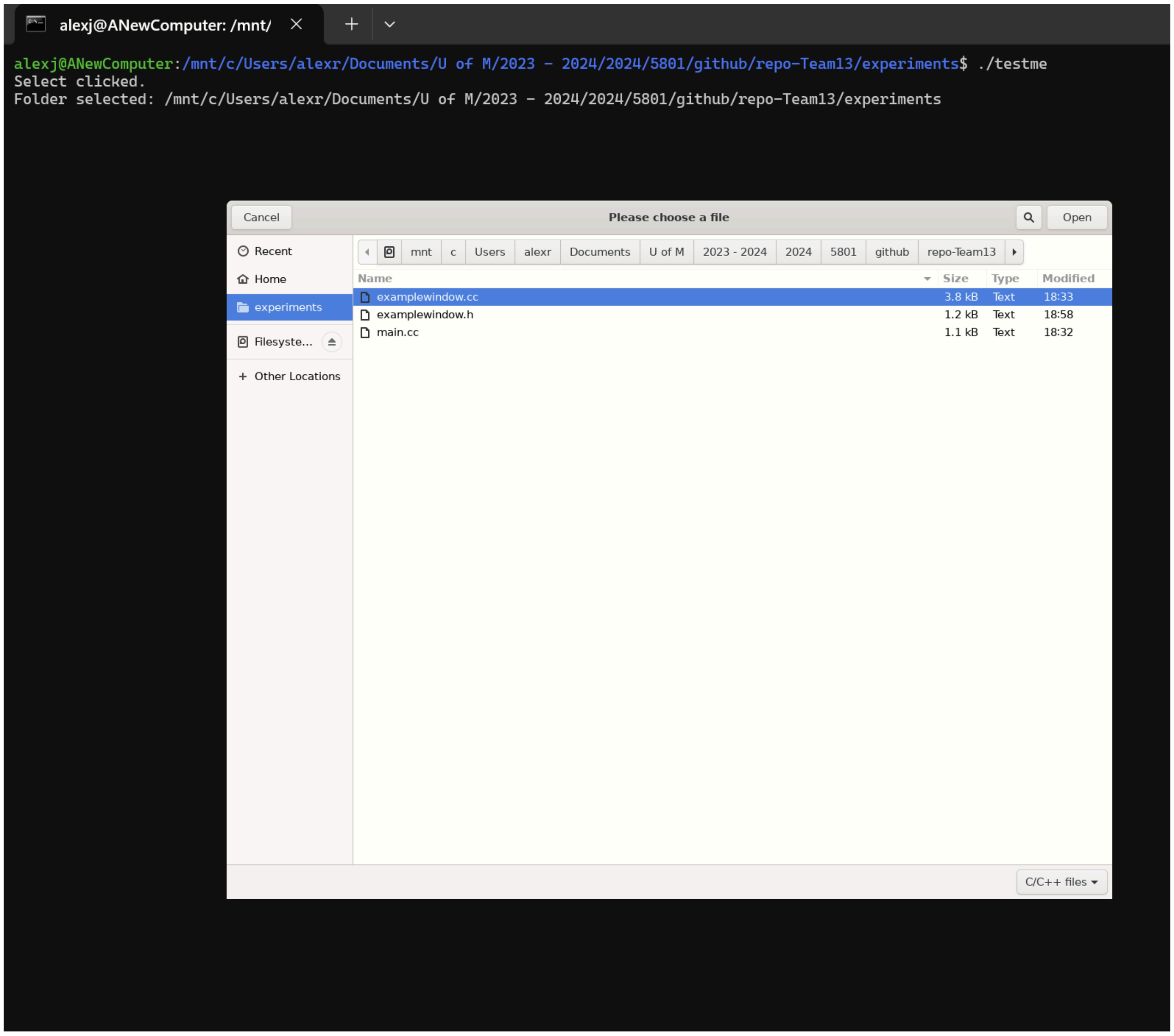
Output: The `Electable` class object that wins the tiebreaker.

6. Human Interface Design

6.1 Overview of User Interface

The user should start this system from the command line or by directly invoking the compiled application. Should the user start the system from the command line, the user may provide a file name, to be consumed as the ballot.csv, as an argument. Should the file name cause an error or not be provided, a file selection GUI will appear. This GUI shall appear until closed or a valid file is submitted.

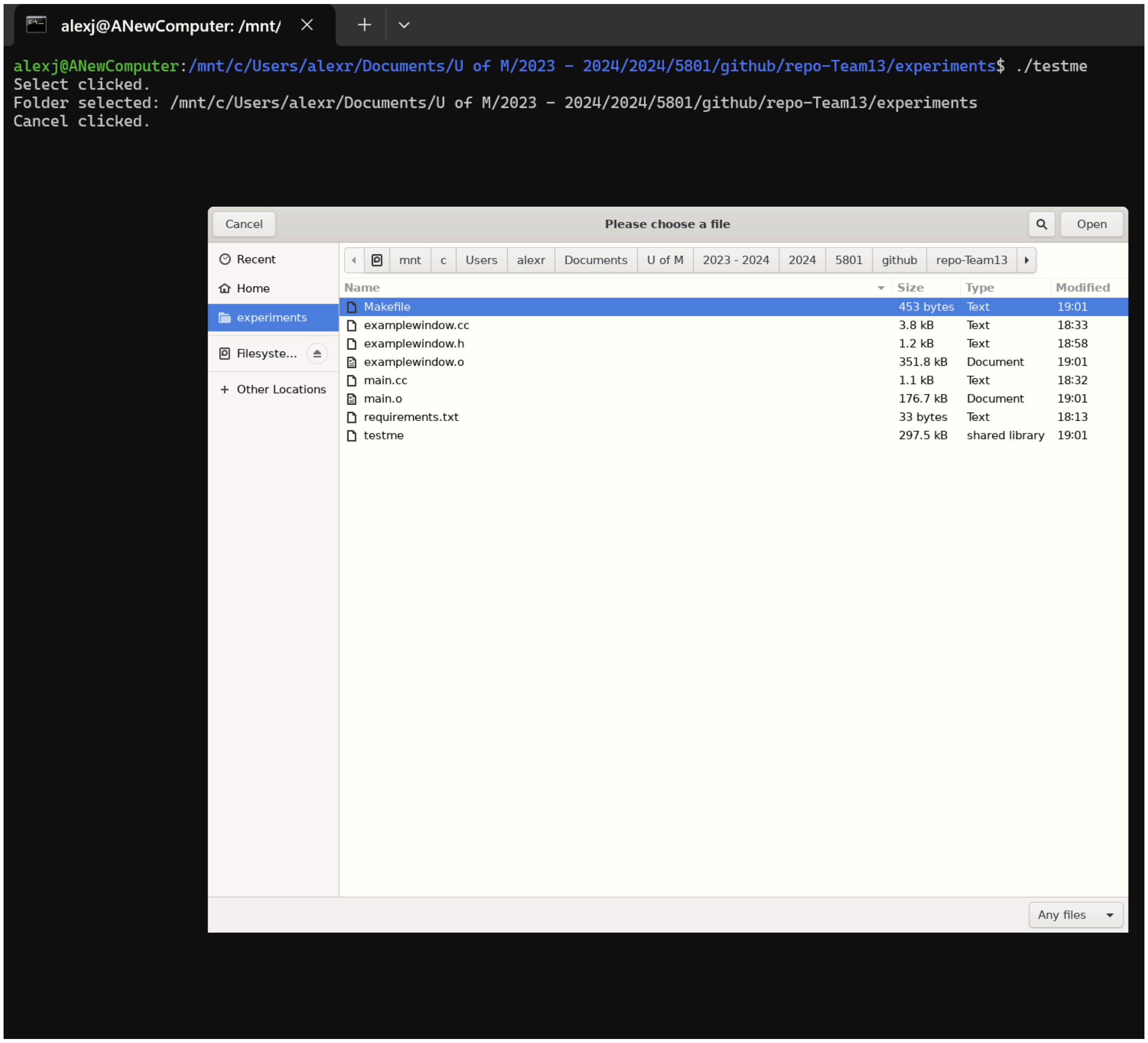
6.2 Screen Images



```
alexj@ANewComputer:/mnt/c/Users/alexr/Documents/U of M/2023 - 2024/2024/5801/github/repo-Team13/experiments$ ./testme
Select clicked.
Folder selected: /mnt/c/Users/alexr/Documents/U of M/2023 - 2024/2024/5801/github/repo-Team13/experiments
```

Name	Size	Type	Modified
examplewindow.cc	3.8 kB	Text	18:33
examplewindow.h	1.2 kB	Text	18:58
main.cc	1.1 kB	Text	18:32

Figure 6.2.1. Selecting only specific file types



```
alexj@ANewComputer:/mnt/c/Users/alexr/Documents/U of M/2023 - 2024/2024/5801/github/repo-Team13/experiments$ ./testme
Select clicked.
Folder selected: /mnt/c/Users/alexr/Documents/U of M/2023 - 2024/2024/5801/github/repo-Team13/experiments
Cancel clicked.
```

Name	Size	Type	Modified
Makefile	453 bytes	Text	19:01
examplewindow.cc	3.8 kB	Text	18:33
examplewindow.h	1.2 kB	Text	18:58
examplewindow.o	351.8 kB	Document	19:01
main.cc	1.1 kB	Text	18:32
main.o	176.7 kB	Document	19:01
requirements.txt	33 bytes	Text	18:13
testme	297.5 kB	shared library	19:01

Figure 6.2.2. Selecting any file type

6.3 Screen Objects and Actions

The upper left corner features a `Cancel` button, canceling file selection and stopping the application, closing this GUI in the process. Also within the window header is a search icon, allowing the user to search for files by name, and an `Open` button, submitting the selected file to the system and closing this GUI. The final feature of the window header is a customizable prompt, "Please choose a file". This text may be changed to indicate file type or give the user additional information. In the bottom right corner is a dropdown menu, here used to discriminate between `C/C++ files` and `Any files`. The system will allow only .csv files. Both the left and upper menus provide useful context, allowing the user to easily navigate and find the desired file. Clicking a tile, for example `Home` or `U of M`, will automatically navigate to that directory. The arrows may be used to view overflowing directories, such as `Experiments`. A right context menu gives information about the displayed files and directories, as well as allowing sorting by toggling the title row elements of each column. Selected objects are highlighted in blue.

7. Requirements Matrix

Use Case	Related Components
Case 1: Audit File Output Actors: Election Official, Voting System	Data: The audit file is created by writeElectable() in the Audit class. Stimulus: An object is passed to the Audit class that is able to record the relevant information and a summary. Response: The audit file is created, and is outputted for the Election Officials to view.
Case 2: Display Election Information Actors: User, Election Official, Audit File	Data: The election information is displayed with the display() function in the Election class Stimulus: Once the election has been conducted, and all the results have been calculated, the Audit file is passed to the Election class where the information will be displayed for the user or election official to see. Response: The election results are printed to the terminal.
Case 3: Get File Header Actors: Voting System	Data: The file header is gotten by getHeader() in the RawData class. Stimulus: Once the file is read by Response: The file header is sent to Raw
Case 4: Read File Actors: Voting System, Election Official	Data: The reading of the input file is done by the read() function in the FileOps class. Stimulus: Once the file is opened, it has to be read Response: The read input file's information is read and sent to the Voting System in order to continue the election.
Case 5: Process CSV File Actors: Voting System	Data: Processing the CSV file is handled by the parseFile() function in the RawData class. Stimulus: Once the file is read by the FileOps class, it is passed to the RawData class where it is processed.

	Response: After the CSV file is processed, its sent to the Voting System for results to be calculated.
Case 6: Resolve Ties Actors: Voting System and candidates in tie.	Data: Resolving ties are resolved by the resolveTie() function in the Voting System class. Stimulus: If there is a tie of two or more candidates during an election, this use case will resolve the tie. Response: A tiebreaker/winner is determined and the results are sent back to the class that requested it.
Case 7: CPL Algorithm Actors: Voting System	Data: Closed Party List is handled by the calculateResults() function in the CPL class. Stimulus: The file header indicates that the election is a Closed Party Election Response: The results of the CPL election are returned
Case 8: OPL Algorithm Actors: Voting System	Data: Open Party List is handled by the calculateResults() function in the OPL class. Stimulus: The file header indicates that the election is an Open Party Election Response: The results of the OPL election are returned
Case 9: Get file name via command line Actors: Users, Election Officials	Data: Getting the file name via command line is done by parseFile() function in the FileOps class. Stimulus: The file is passed by input of the user/election official into the command line, and the file exists and is valid. Response: The file is parsed and sent to the relevant classes for further processes in the election.
Case 10: Get file name via GUI Actors: Users, Election Officials	Data: Getting the file name via GUI is done by parseFile() function in the FileOps class Stimulus: The file is passed by input of the user/election official, but the file name/format is invalid, so the GUI is opened to easily input the correct file. Response: The file is parsed and sent to the relevant classes for further processes in the

	election.